Sergio David Sierra - 201914519

Desarrollo Taller 1 Infraestructura Computacional – Concurrencia:

Parte A:

1. Complete la la siguiente tabla, con respecto a la creación de threads usando la extensión de la clase Thread y la implementación de la interface Runnable.

Se parecen	**Se diferencian**
- Las dos necesitan un método main para ejecutarse. - Las dos deben ejecutar el método start de la clase Thread para empezar a ejecutarse. - Las dos deben sobrescribir el método run para que se ejecuten en simultaneo.	- La versión que usa la clase Thread, debe extender, es decir heredar todas las funciones que la clase Thread tiene. - La versión que usa la interfaz Runnable, puede solamente crear instancias de objetos Thread, ya que como tal es una interfaz y no tienen definidos las funciones que implementa (Métodos abstractos). Por esto último es que se hace el cast de Thread antes de instanciar la clase que se está creando puesto que esta implementa la interfaz Runnable. - La forma sencilla de entenderlo es que cuando se extiende de Thread, finalmente estás haciendo que tu objeto tenga las características de un Thread además de otras que puede tener, mientras que, si implementas la interfaz Runnable, estas creando un objeto Thread a partir de los métodos abstractos que tiene la Interfaz (Siendo mediador la clase creada).

Ejercicio:

Dentro de la carpeta, se encuentra el archivo Taller1Threads.java, para ejecutarlo por entrada estándar, abrir una consola en la ruta del archivo y ejecutar el comando "*javac Taller1Threads.java*" y luego ejecutar el comando "*java Taller1Threads X Y Z*" donde X es el límite superior para el algoritmo, Y son los milisegundos que tardará el Thread1 en ejecutar la siguiente acción y Z los milisegundos que tardará el Thread 2 en ejecutar la siguiente acción. Se recomienda para probar los números en orden, ejecutar el comando "*java Taller1Threads 10 490 500*". Respuesta en consola:

Parte B:

Parte 1:

1. ¿Al ejecutar el programa, el resultado corresponde al valor esperado?

Sí, el programa incrementa un contador 1000 veces el valor de 10000, por lo que al final la variable se imprime con 10,000,000.

2. ¿Al ejecutar el programa, el resultado corresponde al valor esperado?}

Al ejecutar el programa este no funciona correctamente puesto que varios threads hacen acceso a la variable contador al mismo tiempo, haciendo que se modifique en algunas ocasiones de manera incorrecta por la concurrencia.

3. Ejecute cinco veces el programa y escriba el resultado obtenido en cada ejecución.

Ejecución	**Valor obtenido**
1	9147510
2	9138578
3	8739652
4	8853047
5	8839898

4. ¿Hay acceso concurrente a alguna variable compartida? Si es así, diga en dónde.

Si, hay acceso concurrente a la variable contador.

Parte 2:

1. Ejecute cinco veces el programa y escriba el resultado obtenido en cada ejecución.

Ejecución	**Valor obtenido**	**Valor esperado**
1	85353	100293
2	48408	48408
3	90040	90040

4	97481	97481
5	95692	95692

2. ¿Hay acceso concurrente a alguna variable compartida? Si es así, diga en dónde.

Si hay acceso concurrente a la variable mayor. Por ello en algunos casos no encuentra el mayor el problema, aunque casi siempre lo haga.

3. ¿Puede obtener alguna conclusión?

Hay muchos casos en los que la programación concurrente con el uso de varios threads es bastante útil, pero se debe tener siempre en cuenta la manipulación de los atributos globales los cuales son accedidos por los diferentes Threads, ya que puede haber inconsistencias al no cumplir con los requisitos ACID. (Atomicity, Consistency, Isolation and Durability), especificando los requisitos de atomicidad, consistencia y aislamiento.